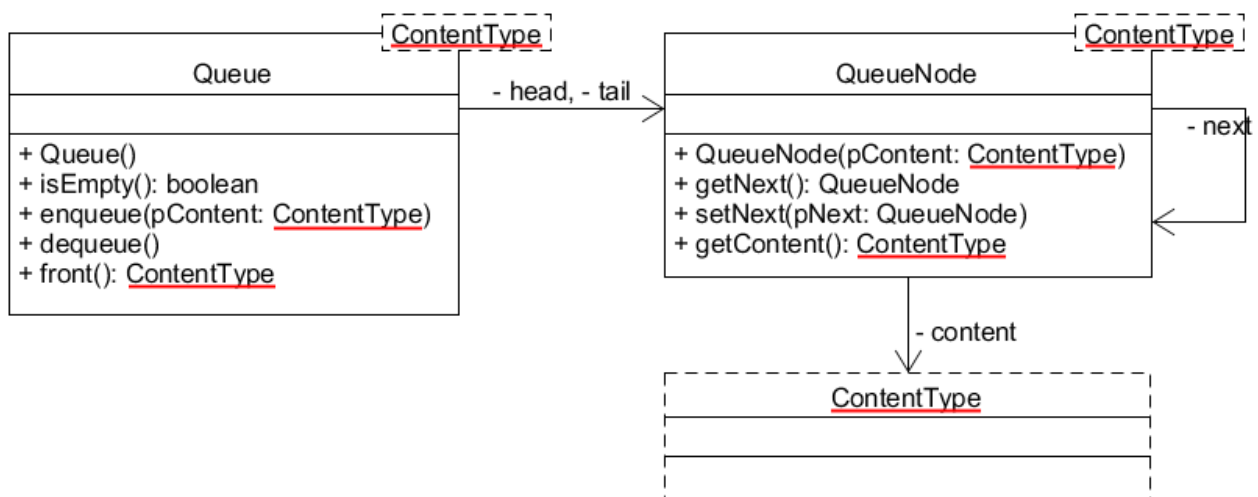


Das Schulministerium stellt Java-Klassen für die Arbeit mit Datenstrukturen (Queue, Stack, Liste usw.) zur Verfügung.

Das Klassendiagramm zeigt eine Warteschlange für Patienten:

- Das **Queue**-Objekt implementiert die Warteschlange.
- Der Anfang heißt **head** (Kopf), das Ende **tail** (Schwanz).
- Jedes Objekt in der Warteschlange wird an ein **QueueNode**- (Knoten-) Objekt gehängt.
- Die Referenz eines Knotens auf seinen Nachfolger heißt **next**.
- Jeder Knoten hat in diesem Beispiel eine Referenz auf ein Objekt der Klasse Patient. Da man eine Queue auch für andere Dinge, z.B. Vokabel-Lernkarten, Netzwerkpakete etc. verwenden kann, wird diese Referenz allgemein **content** genannt (engl. für Inhalt).

## Das vollständige Klassendiagramm der Queue

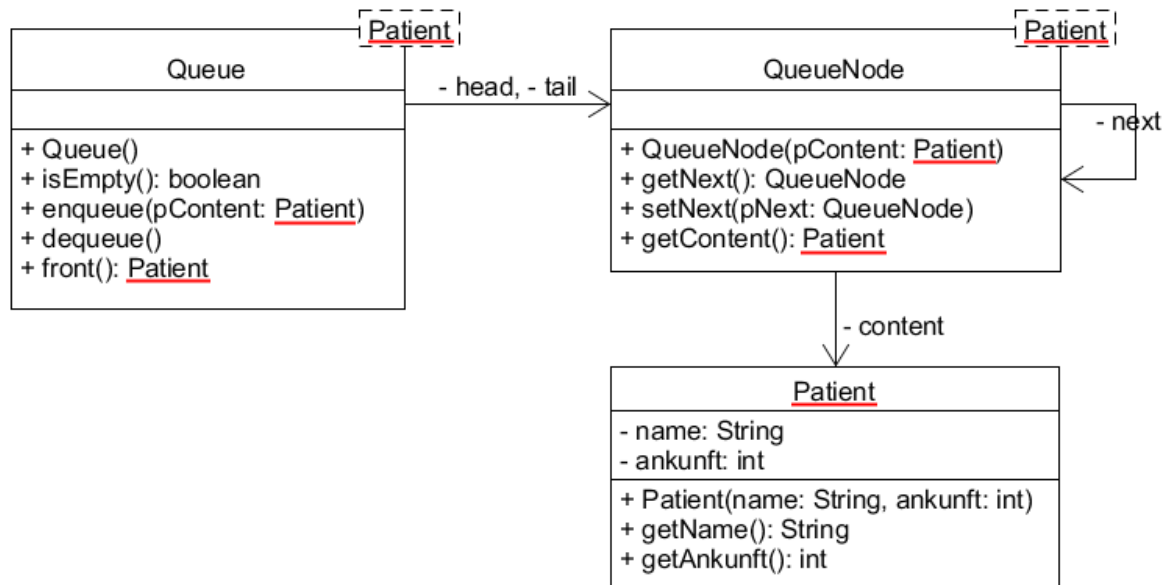


Die vorliegende Klasse für die Queue ist **generisch**: man kann sie für beliebige Inhalts-Objekte verwenden (z.B. Patienten, Callcenter-Anrufe, Netzwerkpakete usw.) Für generische Klassen wie die Queue kann man den Platzhalter „ContentType“ durch jede beliebige Klasse ersetzen:

Man deklariert im Quellcode z.B. ein Objekt `Queue<Patient> q1`, und kann `q1` dann (nur) für Patienten-Objekte verwenden. Dabei wird „ContentType“ an allen Stellen (im Diagramm rot unterstrichen) durch die Klasse „Patient“ ersetzt.

Deklariert man ein Objekt `Queue<Netzwerkpaket> q2`, nimmt es entsprechend Objekte der Klasse `Netzwerkpaket` auf, `ContentType` wird dann durch die Klasse `Netzwerkpaket` ersetzt.

Das folgende Diagramm stellt die Klassen dar, nachdem durch die Deklaration `Queue<Patient>` der Platzhalter „Contentype“ durch die Klasse „Patient“ ersetzt wurde:



## Die Methoden der Klasse „Queue“ (Beispiel: ContentType = Patient)

Für die Programmierung mit der Queue benötigen wir nur die Methoden der Klasse „Queue“. Die Methoden der Klasse „QueueNode“ sind für uns nicht weiter wichtig, sie werden nur für die Implementierung der Klasse Queue benötigt.

- **Queue()** (Konstruktor)  
Erzeugt eine leere Warteschlange. head und tail sind null.
- **boolean isEmpty()**  
Gibt true zurück, falls die Queue leer ist.  
Die Queue ist nicht leer, falls sie mindestens ein Inhaltsobjekt (hier: einen Patienten) enthält.
- **void enqueue(Patient pContent)**  
Ein Patient-Objekt wird ans Ende der Queue eingefügt. Dabei wird ein neues QueueNode-Objekt erzeugt und an den bisher letzten QueueNode angehängt.  
Die Referenz „tail“ wird auf den gerade neu eingefügten QueueNode verschoben.  
(enqueue: englisch für „anstellen“)
- **void dequeue()**  
Das Patient-Objekt am Anfang der Schlange wird entfernt.  
Die Referenz „head“ wird auf den nächsten QueueNode in der Reihe verschoben  
(falls es nur diesen einen Patienten in der Schlange gab, sind head und tail anschließend null).  
Falls es keine Referenz mehr auf den entfernten Patienten gibt, wird das Objekt gelöscht.
- **Patient front()**  
Gibt eine Referenz auf das Patienten-Objekt am Anfang der Schlange zurück.  
D.h. man kann die Queue jederzeit „fragen“, wer gerade am Anfang steht  
(bevor man diesen Patient dann mit dequeue aus der Queue entfernt).  
Falls die Queue leer ist, wird null zurückgegeben.

## Aufgabe

Jogi Löw, Jürgen Klinsmann und Franz Beckenbauer gehen (in dieser Reihenfolge) zum Arzt. Die Situation wird durch ein Programm simuliert, in dem eine Queue verwendet wird.

```
01 public class Wartezimmer
02 {
03     private Queue<Patient> queue;
04
05     public WartezimmerSimulation()
06     {
07         queue = new Queue();                // 1. Diagramm
08     }
09
10     public void main()
11     {
12         Patient p;
13
14         // Zwei neue Patienten stellen sich an
15         p = new Patient("Jogi Löw", 1);
16         queue.enqueue(p);
17         p = new Patient("Jürgen Klinsmann", 2);
18         queue.enqueue(p);                    // 2. Diagramm
19
20         // Der Patient am Anfang der Schlange wird behandelt
21         if (!queue.isEmpty())
22         {
23             p = queue.front();
24             queue.dequeue();
25             Console.println(p.getName() + " wird jetzt behandelt!");
26         }
27
28         // Ein weiterer Patient stellt sich an
29         p = new Patient("Franz Beckenbauer", 3);
30         queue.enqueue(p);
31
32         // Der Patient am Anfang der Schlange wird behandelt
33         if (!queue.isEmpty())
34         {
35             p = queue.front();
36             queue.dequeue();                    // 3. Diagramm
37             Console.println(p.getName() + " wird jetzt behandelt!");
38         }
39     }
40 }
```

- Zeichne je ein Objektdiagramm (insgesamt 3 Stück) aller Objekte des Programms zu den angegebenen Zeitpunkten, und zwar 1. nach Zeile 07, 2. nach Zeile 18, 3. nach Zeile 36
- Nach dem Aufruf von `dequeue()` in Zeile 24 wird das Objekt für den ersten Patienten nicht gleich gelöscht. Warum?
- Gib an, welche Ausgabe die `main`-Methode auf der Konsole macht.